R : Un linguaggio per l'analisi statistica e la rappresentazione grafica di dati (Prima parte)

Dipartimento di Scienze Politiche e Sociali -
Corso di Analisi dei Dati

25 ottobre 2018

Indice

Capitolo 1

Introduzione a R

1.1 R ed R Studio

R è un programma per la grafica e l'analisi statistica. R è largamente diffuso in virtù di diverse caratteristiche vantaggiose, alcune delle quali sono elencate di seguito:

- R è un programma di libera diffusione disponibile per tutti i principali sistemi operativi (Unix, Windows, Mac OS) alla pagina web

  https://www.r-project.org

- esso fornisce un insieme piuttosto ampio di funzioni statistiche per l'analisi dei dati e le elaborazioni grafiche;
- è basato sul linguaggio *S*, linguaggio e ambiente di sviluppo di alto livello per l'analisi dei dati e la grafica;
- dispone di applicazioni per la grafica estremamente potenti che possono essere facilmente adattate ai dati a disposizione;
- è un potente linguaggio ad oggetti che può essere facilmente esteso dall'utente (si rendono continuamente disponibili nuovi pacchetti, al momento sono 12751).

All'avvio appare semplicemente come una "console" con un cursore, rappresentato dal simbolo '>', dove digitare i comandi e le istruzioni, che vengono eseguite dando l'invio. È spesso utile memorizzare i comandi e le funzioni utilizzate in un file di estensione .R (*script*), per poterle riutilizzare in seguito.
Le elaborazioni in R proposte nel corso verranno illustrate utilizzando R Studio, un ambiente di sviluppo integrato per R , pensato per facilitarne l'utilizzo e disponibile al sito http://www.rstudio.com. R studio consente di integrare diverse funzionalità che sono estremamente utili, soprattutto se si utilizza R per progetti più complessi. Si presenta suddiviso in

- finestra del codice: in quest'area è possibile aprire, creare e scrivere i vostri script;

- finestra della console: questa zona è la console R effettiva in cui vengono eseguiti i comandi e visualizzato l'output;
- finestra degli oggetti: in quest'area è possibile trovare una lista di tutti gli oggetti creati nello spazio di lavoro in cui si sta lavorando;
- la finestra dei pacchetti-dei grafici-dell'aiuto in linea: in quest'area è possibile caricare facilmente i pacchetti, aprire file di aiuto R e visualizzare i grafici.

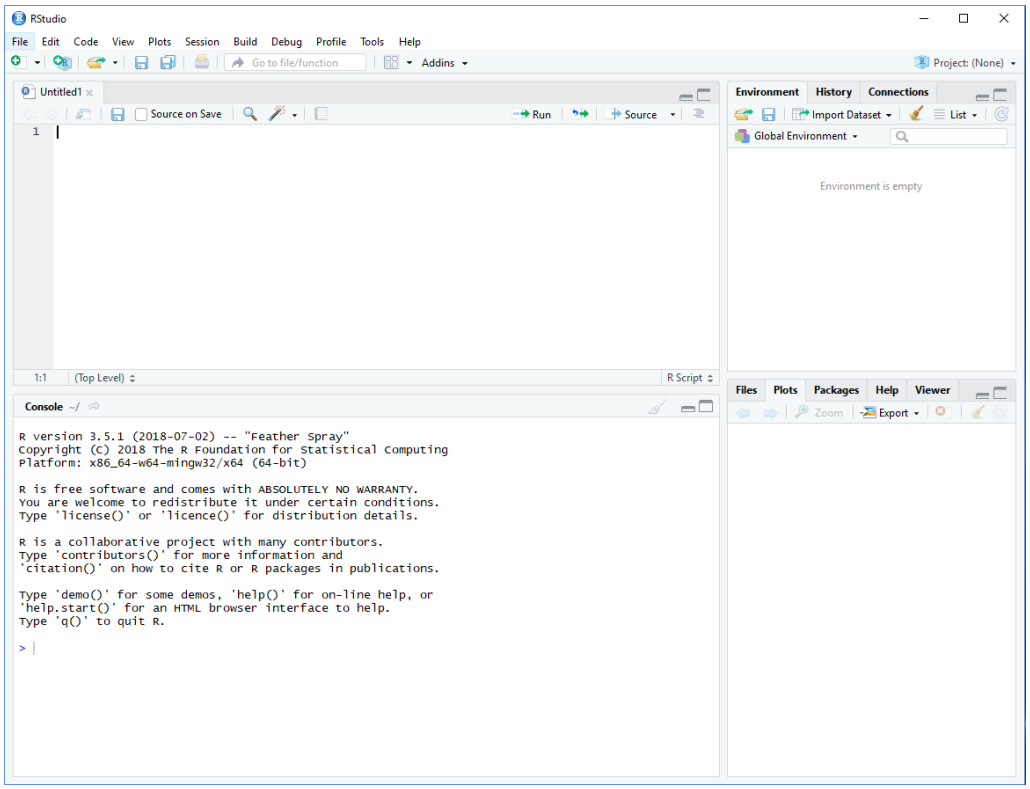


Figura 1.1: L'interfaccia grafica di R Studio.

1.1.1 Spazio di lavoro, aiuto in linea e documentazione

Dopo aver avviato R Studio, è importante impostare prima la *directory*, ovvero la cartella di lavoro, che contiene i dati che si intende utilizzare. In R Studio occorre selezionare 'Session' e poi 'Set Working Directory' dal menu, infine selezionare la cartella desiderata.
Le librerie (*packages*), consentono di eseguire elaborazioni statistiche e calcoli matematici, anche avanzati. Un package è un insieme di funzioni di norma destinato ad una specifica applicazione. Alcuni packages sono già installati in R ed è sufficiente caricarli. Per gli altri, è necessario scaricarli dai server di R ed installarli. Si vedrà in seguito come farlo.
Come accennato, R fornisce un sistema di *help* utile, ad esempio, per sapere come opera una funzione. È possibile accedere alla finestra di aiuto mediante il comando

```
help(nome.funzione)
```

oppure

```
?(nome.funzione)
```

o ancora, dalla apposita finestra (in basso a destra) in R Studio.
R è orientato agli oggetti, nel senso che possiamo creare entità denominate *oggetti*. Un **oggetto** è qualsiasi cosa a cui possiamo dare un nome. Gli oggetti possono essere di diversi tipi: numeri, vettori, matrici, o strutture più complesse, come i dataframes e le liste. Una volta creati, essi vengono salvati in un'area dedicata, chiamata *workspace*. È possibile consultare la lista degli oggetti presenti nell'ambiente di lavoro utilizzando l'apposita finestra "Environment" in alto a destra in R Studio (si veda **??**), oppure digitando il seguente comando dalla console:

```
ls()
```

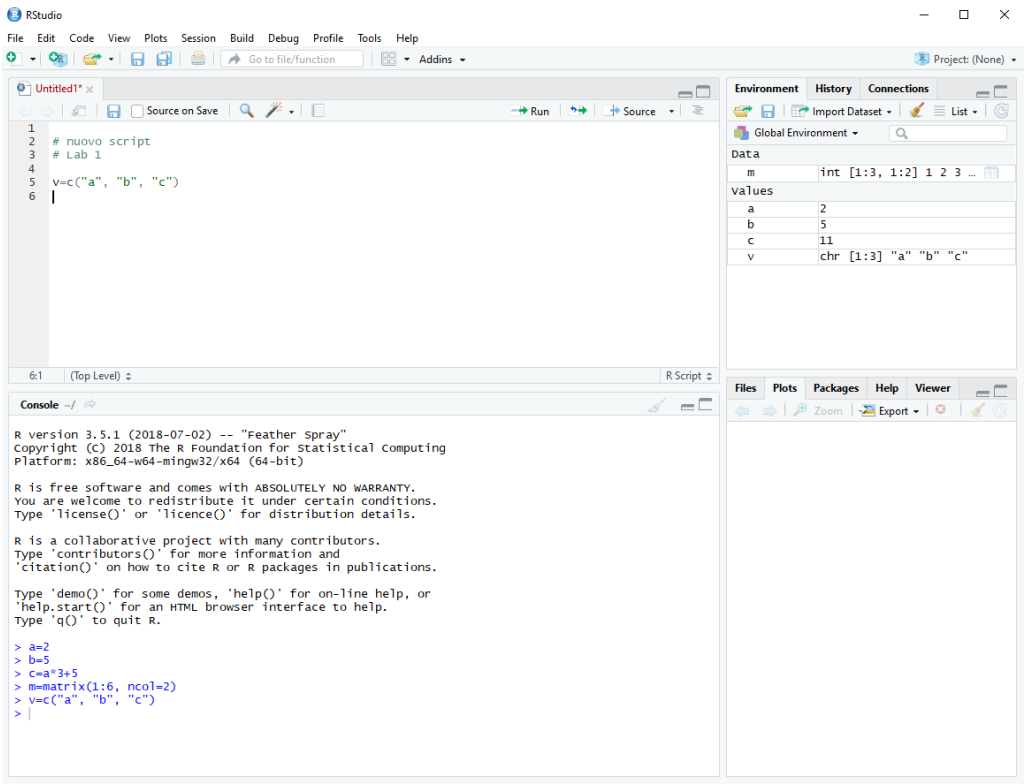


Figura 1.2: Spazio di lavoro e oggetti in R Studio.

La funzione `rm()` può essere usata per eliminare uno o più oggetti dall'ambiente di lavoro; ad esempio gli oggetti chiamati `a` e `b` presenti nel workspace, possono essere eliminati tramite il comando

```
rm(a, b)
```

Quando si inizia una nuova sessione è consigliabile pulire il *workspace*, eliminando tutti gli oggetti già presenti; è facile eseguire questa operazione tramite il menu 'Session' > 'Clear Workspace'.
Al termine della sessione viene richiesto se si vuole salvare la sessione di lavoro come file '.RData' nella *directory* corrente. Tale operazione può essere effettuata selezionando dal menu 'Session' > 'Save Workspace As...'. Ciò è particolarmente utile nel caso fosse necessario riutilizzare gli oggetti creati in un lavoro futuro. Infatti, il caricamento ('Session' > 'Load Workspace...') di uno spazio di lavoro salvato in precedenza consente di trovare di nuovo tutti gli oggetti creati nella sessione salvata.

1.2 I concetti di base

Prima di cominciare è importante sapere che:

- R è "case sensitive" cioè considera come caratteri differenti le lettere maiuscole e minuscole, quindi A e a sono ad esempio nomi di variabili diverse;
- tutte le lettere ed i numeri possono essere utilizzati per dare nomi a variabili, funzioni e procedure; sono però esclusi gli spazi e tutti i segni di punteggiatura ad eccezione del punto ('.'), comunemente usato per separare le parole che compongono il nome; ogni nome deve cominciare con una lettera e non con un numero;
- ogni comando deve essere separato da un punto e virgola (';') oppure deve essere su una nuova linea;
- testi di commento possono essere aggiunti dopo il simbolo cancelletto ('#'), tutto ciò che si trova dopo questo simbolo e fino alla riga successiva viene ignorato da R;
- se si va a capo prima della conclusione di un comando R mostrerà all'inizio della riga il simbolo più ('+');
- è possibile richiamare i comandi impartiti in precedenza utilizzando la freccia ùu' della tastiera.

R è in grado di gestire con la stessa disinvoltura numeri reali, numeri complessi, stringhe di testo, e valori logici. I numeri complessi si distinguono per la presenza della parte immaginaria, che deve sempre essere esplicitata, ad esempio:

```
sqrt(-17+0i)

## [1] 0+4.123106i
```

Le stringhe, cioè insiemi di caratteri, sono racchiuse da doppie virgolette. I valori logici disponibili sono `TRUE` e `FALSE`.

1.2.1 L'aritmetica con R

I comandi di base possono essere sia espressioni che assegnazioni. Se il comando è un'espressione, R restituisce il risultato, come illustrato negli esempi seguenti

```
12 > 10

## [1] TRUE

1+2+3

## [1] 6

2+3*4

## [1] 14

3/2+1

## [1] 2.5

2+(3*4)

## [1] 14

(2 + 3) * 4

## [1] 20

4*3^3

## [1] 108
```

Si noti che le operazioni sono eseguite rispettando il classico ordine di priorità salvo che questo non venga modificato utilizzando le parentesi.
Tutte le funzioni matematiche elementari sono presenti in R come funzioni di base. Alcune di esse sono elencate nella tabella seguente:

nome funzione in R	funzione
`sqrt`	radice quadrata
`abs`	valore assoluto
`sum, mean, var`	somma, media, varianza
`max, min`	massimo e minimo
`exp,log`	esponenziale e logaritmo

Un'assegnazione, invece, valuta un'espressione e salva il risultato in uno specifico oggetto.

L'assegnazione viene fatta mediante il simbolo `<-`, oppure il simbolo `=`. È anche possibile effettuare l'assegnazione da sinistra a destra, utilizzando il simbolo `->`.
Il risultato di un'assegnazione non viene immediatamente visualizzato. L'utente può visualizzarlo digitando il nome dell'oggetto. L'oggetto creato tramite assegnazione può essere riutilizzato in espressioni o in altre assegnazioni, come mostrato negli esempi seguenti.

```
x <- sqrt(2)
x

## [1] 1.414214

x*3

## [1] 4.242641

x^3 -> y
y

## [1] 2.828427

z = exp(x)
z

## [1] 4.11325

w = log(y)+x
w

## [1] 2.453934
```

Si noti che, se si assegna allo stesso oggetto più di un valore, R memorizza l'ultimo valore attribuito.
Gli operatori logici utilizzabili in R sono:

simbolo	operazione
`<`	minore
`<=`	minore o uguale
`>`	maggiore
`>=`	maggiore o uguale
`==`	uguale
`!=`	diverso
`&`	intersezione ('e')
`|`	l'unione ('o')
`!`	negazione ('non')

Essi restituiscono un valore logico, che può essere sia `TRUE` che `FALSE`. Di seguito alcuni esempi:

```
x

## [1] 1.414214

x > 10

## [1] FALSE

x <= 10

## [1] TRUE

tf = x > 10
tf

## [1] FALSE
```

```
5!=5

## [1] FALSE
```

Da notare che risulta

```
TRUE==1

## [1] TRUE

FALSE==0

## [1] TRUE
```

Le operazioni aritmetiche "impossibili", come ad esempio zero diviso zero restituiscono il valore `NaN` che significa che "non è un numero".

1.2.2 I principali oggetti

Alla base della programmazione in R vi sono gli oggetti. La classe (class) definisce il tipo di oggetto, ad esempio, vettore, matrice, lista, ecc. I principali oggetti sono descritti di seguito.

scalari	contengono un singolo valore
vettori	insieme lineare di elementi omogenei per tipologia, tutti numeri, tutte stringhe, ecc.
fattori	vettore utilizzato per classificare o suddividere in "livelli" gli elementi di un altro vettore
array e matrici	sono vettori multidimensionali, le matrici hanno due dimensioni: righe e colonne
liste	insieme di elementi disomogenei tra loro
data frame	sono matrici bidimensionali dove ogni colonna può avere un tipo di dato diverso dalle altre

Il modo di un oggetto definisce invece il tipo di dato contenuto nell'oggetto. Come accennato in precedenza, R consente di manipolare dati di tipo numerico, logico, caratteri e stringhe. La maggior parte delle funzioni di R accetta come argomento oggetti di diverse classi e modi. In tal caso, il comportamento della funzione dipende dalle caratteristiche dell'oggetto a cui viene applicata. Il comando `mode()` restituisce il tipo di dato contenuto nell'oggetto, ad esempio *logical* (`TRUE` o `FALSE`), *character* o *numeric*, come nell'esempio riportato di seguito:

```
y = "Ciao"
mode(y)

## [1] "character"
```

1.2.3 Variabili e vettori

È possibile creare una **variabile** con una assegnazione come negli esempi visti in precedenza:

```
nome.variabile = valore
```

`nome.variabile` può essere un qualunque nome a scelta mentre `valore` può essere un valore numerico, un testo racchiuso da doppie virgolette, il nome di un'altra variabile, il risultato di una operazione o di una qualunque funzione.
Un **vettore** è un insieme di dati omogenei, cioè dello stesso tipo. La funzione `c()` combina i valori presenti nel suo elenco, separati da virgola, in un vettore. La creazione del vettore `x` si ottiene, ad esempio, nel modo seguente:

```
x = c(10, 15, 6.4, 3, 18) #assegnazione del vettore
x                         #visualizzazione del contenuto del vettore

## [1] 10.0 15.0  6.4  3.0 18.0

str(x)                    #restituisce la struttura dell'oggetto

##  num [1:5] 10 15 6.4 3 18
```

Il numero uno tra parentesi quadre ('[1]') che indica che la riga mostra un insieme di dati a cominciare dal primo: se per mostrare tutti gli elementi occorrono più righe, all'inizio di ciascuna viene indicato il numero dell'elemento con cui la riga comincia. Si noti l'utilizzo del comando `str()`, utile per ottenere informazioni sulla struttura interna di un qualsiasi oggetto.
Con R è possibile creare facilmente delle sequenze regolari di numeri, come nei seguenti esempi:

```
1:18                        #genera un vettore dei numeri da 1 a 18

##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18

seq(from=1, to=18, by=1.5) #vettore da 1 a 18 con intervalli di 1.5

##  [1]  1.0  2.5  4.0  5.5  7.0  8.5 10.0 11.5 13.0 14.5 16.0 17.5
```

L'ultima riga di comando può essere sostituita dalla forma concisa `seq(1, 18, 1.5)`.
È anche possibile creare vettori con elementi ripetuti mediante il comando `rep()`:

```
rep(1,15) #genera un vettore in cui si ripete 15 volte il valore 1

##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

a = c(rep(2,3),4,5,rep(1,5),11:15)
a

##  [1]  2  2  2  4  5  1  1  1  1  1 11 12 13 14 15
```

Un sottoinsieme di elementi di `x` può essere selezionato utilizzando le parentesi quadre dopo il nome del vettore e indicando l'indice (o gli indici) corrispondente all'elemento (agli elementi) da selezionare. Ad esempio:

```
y = x[2]         #seleziona il secondo elemento di x
y

## [1] 15

z = x[2:4]       #seleziona gli elementi dal secondo al quarto
z

## [1] 15.0  6.4  3.0

w = x[c(2, 5)]  #seleziona il secondo e il quinto elemento di x
w

## [1] 15 18
```

Quando gli indici in parentesi quadre sono preceduti dal segno '-', gli elementi corrispondenti vengono eliminati dal vettore (non dal vettore originale ma nella selezione):

```
x = c(1, 2, 4, 8, 16, 32)
x

## [1]  1  2  4  8 16 32

x[-4]

## [1]  1  2  4 16 32
```

Ai vettori possono essere facilmente applicate l'aritmetica di base e gli operatori logici:

```
x = 1:10

x*2

##  [1]  2  4  6  8 10 12 14 16 18 20

x > 5

##  [1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Quando si compiono operazioni tra due vettori, il primo elemento dell'uno viene combinato con il primo elemento dell'altro, il secondo col secondo e così via. Se un vettore è più corto dell'altro, quando termina viene riutilizzato dall'inizio.

```
x = rep(10,8)
y = c(1,2)
x

## [1] 10 10 10 10 10 10 10 10

y

## [1] 1 2

x*y

## [1] 10 20 10 20 10 20 10 20

x+y

## [1] 11 12 11 12 11 12 11 12
```

Altre operazioni utili con i vettori sono illustrate di seguito:

```
z = c(72.6, 40.1, 59.6, 92.3, 65.4, 27.1)

sort(z)      #valori in ordine crescente

## [1] 27.1 40.1 59.6 65.4 72.6 92.3

order(z)     #ordine dei valori nel vettore

## [1] 6 2 3 5 1 4

length(z)    #numero di elementi nel vettore

## [1] 6

range(z)     #range dei valori nel vettore

## [1] 27.1 92.3

mean(z)      #media aritmetica

## [1] 59.51667

var(z)       #varianza dei valori del vettore

## [1] 541.6777
```

1.2.4 Array e matrici

Le **matrici** possono essere create con il comando `matrix()`. Nel caso più semplice, si passano come argomenti un vettore di elementi, e il numero di righe o colonne desiderato:

```
x = matrix(c(2,3,5,7,11,13), nrow=3)
x

##      [,1] [,2]
## [1,]    2    7
## [2,]    3   11
## [3,]    5   13

x = matrix(c(2,3,5,7,11,13), ncol=3)
x

##      [,1] [,2] [,3]
## [1,]    2    5   11
## [2,]    3    7   13
```

È importante osservare che, in assenza di istruzioni diverse da parte dell'utente, le matrici sono costruite inserendo gli elementi per colonna.
Una matrice può essere anche creata importando gli elementi da un file esterno. Per esempio, si consideri il file `ex.data` che contiene i seguenti valori

```
TITLE extra line
2  3  5
11 13 17
```

Il contenuto del file può essere letto e assegnato ad una matrice 2×3 usando i seguenti comandi:

```
X = scan("ex.data", skip = 1, quiet = TRUE)
X

## [1]  2  3  5 11 13 17
```

```
mx = matrix(X, ncol=3, byrow=TRUE)
mx

##      [,1] [,2] [,3]
## [1,]    2    3    5
## [2,]   11   13   17
```

Specificando l'argomento `byrow=TRUE` si richiede che gli elementi vengano inseriti per riga invece che per colonna. La funzione `dim()` restituisce le dimensioni di una matrice:

```
dim(mx)

## [1] 2 3
```

Analogamente a quanto visto nel caso dei vettori, è possibile selezionare alcuni elementi da una matrice usando le parentesi quadre `[]` e specificando gli indici di riga e colonna in questo ordine:

```
mx[2,1]     #seleziona l'elemento nella riga 2 e colonna 1

## [1] 11

mx[2,2]     #seleziona l'elemento nella riga 2 e colonna 2

## [1] 13
```

Per estrarre una intera riga (colonna) si fa semplicemente riferimento all'indice di riga (colonna), come nel seguente esempio:

```
mx[2, ]      #estrae la seconda riga

## [1] 11 13 17

mx[ ,3]      #estrae la terza colonna

## [1]  5 17
```

Infine, definita la matrice `x` come segue

```
x = matrix(1:16, ncol=4)
x

##      [,1] [,2] [,3] [,4]
## [1,]    1    5    9   13
## [2,]    2    6   10   14
## [3,]    3    7   11   15
## [4,]    4    8   12   16
```

otteniamo facilmente la matrice `y` con solo le righe 1 e 4 e le colonne 3 e 4 di `x`:

```
y = x[c(1,4),c(3,4)]
y

##      [,1] [,2]
## [1,]    9   13
## [2,]   12   16
```

Gli **array** sono vettori multidimensionali. L'esempio più semplice è quello delle matrici, cioè array bidimensionali rappresentabili come una tabella con un certo numero di righe e colonne. Il comando `array()` richiede di specificare un vettore di elementi e il numero di elementi per ciascuna dimensione attraverso l'argomento `dim`:

```
x = 1:20
ax = array(x, dim=c(5,2,2))
ax

## , , 1
##
##      [,1] [,2]
## [1,]    1    6
## [2,]    2    7
## [3,]    3    8
## [4,]    4    9
## [5,]    5   10
```

```
##
## , , 2
##
##      [,1] [,2]
## [1,]   11   16
## [2,]   12   17
## [3,]   13   18
## [4,]   14   19
## [5,]   15   20

dim(ax)

## [1] 5 2 2
```

È possibile estrarre elementi da un *array* specificando l'indice su ciascuna dimensione in parentesi quadre:

```
ax[3,2,1]

## [1] 8

ax[, 2, ]

##      [,1] [,2]
## [1,]    6   16
## [2,]    7   17
## [3,]    8   18
## [4,]    9   19
## [5,]   10   20

ax[-1,,1]

##      [,1] [,2]
## [1,]    2    7
## [2,]    3    8
## [3,]    4    9
## [4,]    5   10
```

1.2.5 Altri tipi di oggetti

I vettori sono le strutture più semplici in R , ma esistono strutture piú complesse, come le liste, che consentono di legare assieme dati che non condividono la stessa struttura, come ad esempio matrici, singole stringhe, ecc.
Una **lista** si crea specificando un nome per ciascun elemento. Ogni elemento a sua volta può essere un qualunque tipo di oggetto, ad esempio

```
x1 = 1:3
x1

## [1] 1 2 3

x2 = c("A", "B", "C","D","E")
x2

## [1] "A" "B" "C" "D" "E"

x3 = matrix(1:12,nrow=3)
x3

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

list1 = list(x1, x2, x3)
list1

## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "A" "B" "C" "D" "E"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

L'elemento è identificato da un numero progressivo. Per accedere agli elementi di una lista si utilizzano le doppie parentesi quadre:

```
list1[[1]]

## [1] 1 2 3

list1[[2]]

## [1] "A" "B" "C" "D" "E"

list1[[3]]

##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Una volta che si accede ad uno specifico elemento di una lista è possibile estrarre informazioni in maniera analoga a quanto fatto nel caso di matrici, vettori, arrays e così via.

```
list1[[3]][2,3]

## [1] 8

l1 = list1[[1]]
l1

## [1] 1 2 3

l1[2]

## [1] 2
```

Assegnando un nome a ciascun elemento della lista si può accedere ai diversi elementi abbinando il simbolo $ al nome dell'elemento:

```
list2 = list(comp1 = x1, comp2 = x2, comp3 = x3)
list2

## $comp1
## [1] 1 2 3
##
## $comp2
## [1] "A" "B" "C" "D" "E"
##
## $comp3
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12

list2$comp2

## [1] "A" "B" "C" "D" "E"

list2$comp2[3]

## [1] "C"

list2$comp3[1,]

## [1]  1  4  7 10
```

I nomi dei componenti di una lista possono essere visualizzati attraverso il comando `names()`:

```
names(list2)

## [1] "comp1" "comp2" "comp3"
```

che restituisce `NULL` se i nomi non sono stati assegnati. Infine si osservi che quando la funzione `c()` è usata con liste come argomenti si ottiene ancora una struttura di tipo lista i cui elementi sono quelli delle liste in input concatenati in sequenza:

```
list12=c(list1,list2)
list12

## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "A" "B" "C" "D" "E"
##
## [[3]]
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## $comp1
## [1] 1 2 3
##
## $comp2
## [1] "A" "B" "C" "D" "E"
##
## $comp3
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

Concludiamo questa sezione illustrando brevemente cosa sono e come si manipolano i **fattori** in R . I fattori sono vettori utilizzati per classificare o suddividere in "livelli" gli elementi di un altro vettore di pari lunghezza. In pratica, i fattori sono il modo in cui R rappresenta le variabili categoriali. In particolare, un fattore non ordinato può essere visto come una variabile qualitativa sconnessa, un fattore ordinato come una variabile qualitativa ordinabile.

Si supponga di disporre del Paese di provenienza di 15 lavoratori:

```
country=c("Italy","Germany","France","Germany","Germany","Germany",
          "France","Italy","Italy","France","Italy","Italy","Italy",
          "Italy","Germany")
country
```

```
##  [1] "Italy"   "Germany" "France"  "Germany" "Germany" "Germany"
##  [7] "France"  "Italy"   "Italy"   "France"  "Italy"   "Italy"
## [13] "Italy"   "Italy"   "Germany"
```

Creiamo un fattore utilizzando la funzione `factor()`:

```
countryf = factor(country)
countryf
```

```
##  [1] Italy   Germany France  Germany Germany Germany France  Italy
##  [9] Italy   France  Italy   Italy   Italy   Italy   Germany
## Levels: France Germany Italy
```

Si può notare come la visualizzazione sia leggermente diversa dagli oggetti di tipo vettore visti finora. Per determinare i livelli di un fattore si può utilizzare la funzione `levels()`:

```
levels(countryf)
```

```
## [1] "France"  "Germany" "Italy"
```

I livelli dei fattori sono memorizzati in ordine alfabetico, o nell'ordine specificato al momento in cui sono stati definiti. Talvolta i livelli hanno un ordinamento naturale che vogliamo sia mantenuto nella nostra analisi statistica. La funzione `ordered()` applicata ai livelli dei fattori crea tali fattori ordinati.
Supponiamo di disporre anche dell'età degli stessi lavoratori

```
age = c(47,44,44,40,38,36,42,34,34,44,45,41,28,46,43)
```

Possiamo utilizzare il comando `tapply()` per applicare una funzione, nel nostro caso `mean()`, a ogni gruppo di elementi del primo argomento, nel nostro caso `age` definito dai livelli del secondo argomento, qui `countryf`

```
tapply(age, countryf, mean)
```

```
##   France  Germany    Italy
## 43.33333 40.20000 39.28571
```

Il risultato è una struttura della stessa lunghezza del numero di livelli del fattore. Alla stessa famiglia di `tapply()` appartengono anche le funzioni `apply()`, `lapply()` e `sapply()`. Rimandiamo all'*help* per l'utilizzo specifico di queste funzioni.

1.3 I dati e il data frame

Le operazioni che vedremo di seguito riguardano tutte un particolare tipo di oggetto: il **data frame**. Esso è, in termini pratici, una sorta di contenitore in cui sono conservati elementi di diversa natura con alcune caratteristiche e nel rispetto di alcuni vincoli. Un data frame può essere visto come una matrice le cui righe sono le unità statistiche e le colonne (variabili) possono contenere diversi tipi di dati (numerici, stringhe, ecc.). Più precisamente, un data frame è una lista di classe `data.frame` con le seguenti caratteristiche:

- gli elementi che lo compongono devono essere vettori (numerici, di caratteri o logici o fattori);
- i vettori non numerici sono come opzione predefinita trattati come fattori, e i livelli di tali fattori sono gli unici valori che appaiono nel vettore;
- le strutture di tipo vettore che appaiono come componenti del data frame devono tutte avere la stessa lunghezza, e le matrici lo stesso numero di righe.

Gli oggetti che soddisfano queste condizioni possono formare un data frame, ad esempio, in questo modo:

```
sex = c("F","F","M","F","F","M","F","M","M","M","M","M","F","F","M")
under40 = age < 40
Workers = data.frame(Country=countryf,Age=age, Sex=sex,
                     Under40=under40)
```

Per visualizzare gli elementi di `Workers` che ha struttura `data.frame`

```
str(Workers)

## 'data.frame': 15 obs. of  4 variables:
##  $ Country: Factor w/ 3 levels "France","Germany",..: 3 2 1 2 2 2 1 3 3 1 ...
##  $ Age    : num  47 44 44 40 38 36 42 34 34 44 ...
##  $ Sex    : Factor w/ 2 levels "F","M": 1 1 2 1 1 2 1 2 2 2 ...
##  $ Under40: logi  FALSE FALSE FALSE FALSE TRUE TRUE ...
```

Il comando

```
Workers$Age

##  [1] 47 44 44 40 38 36 42 34 34 44 45 41 28 46 43
```

richiama una variabile (colonna) del data frame, mentre se si vuole eliminare una componente si assegna NULL all'elemento:

```
Workers$Sex = NULL
head(Workers)

##   Country Age Under40
## 1   Italy  47   FALSE
## 2 Germany  44   FALSE
## 3  France  44   FALSE
## 4 Germany  40   FALSE
## 5 Germany  38    TRUE
## 6 Germany  36    TRUE
```

1.3.1 Importazione di dati

Esistono diversi modi per disporre un insieme di dati nella struttura di data frame in R . L'input manuale di dati può essere eseguito mediante il comando `data.frame()` come visto prima.
Se si dispone di un insieme di dati (di solito una matrice dati) in un certo tipo di formato come `.txt`, `.dat`, `.csv`, è possibile importarla in R attraverso la funzione `read.table()`:

```
read.table(<percorso + nome file>, <argomenti>)
```

Si noti che occorre fornire il "percorso" del file. Se lo si omette, R andrà a cercare il file specificato nella funzione all'interno della cartella di lavoro corrente (per verificare il percorso della cartella di lavoro corrente si usa la funzione `getwd()`). Tra gli argomenti della funzione `read.table` i più comunemente usati sono:

- `header=TRUE` per leggere i nomi delle variabili (se presenti) nella prima riga del file;
- `sep` e `dec`, per specificare il carattere che separa le colonne e il separatore decimale, rispettivamente (ad esempio, `sep="\t"` sta a indicare che il separatore dei campi è un tabulatore e `dec="."` indica che il separatore decimale è il punto);
- `row.names=1` per importare gli identificatori di riga all'inizio di ogni riga (se presenti).

Per ottenere ulteriori dettagli si può ricorrere all'*help* digitando `?read.table`.
È utile considerare due istruzioni che consentono di leggere dati da un file di tipo Excel. Se un foglio di lavoro Excel viene salvato in formato `.csv` allora è utile la funzione `read.csv()`

```
read.csv(<percorso + nome file>, <argomenti>)
```

Se invece si vuole leggere direttamente il file di estensione `.xlsx` si può ricorrere alla funzione `read.xlsx()` nel pacchetto `openxlsx`. Utilizziamo questo comando per leggere il file `food_coded_rev2.xlsx` riguardante le abitudini alimentari di 125 studenti della Mercyhurst University (Stati Uniti); i dati sono disponibili al sito https://www.kaggle.com/borapajo/food-choices. Prima di importare i dati è necessario installare il pacchetto `openxlsx`, che dovrà essere caricato mediante la funzione `library()`:

```
install.packages("openxlsx")
```

```
library(openxlsx)
```

È possibile installare i pacchetti in R Studio selezionando l'apposito comando nella finestra dei pacchetti in basso a destra; si aprirà una nuova finestra dove verrà digitato il nome del pacchetto da installare. Successivamente, per caricare il pacchetto è sufficiente spuntare l'icona corrispondente nell'elenco dei pacchetti disponibili (si veda **??**).

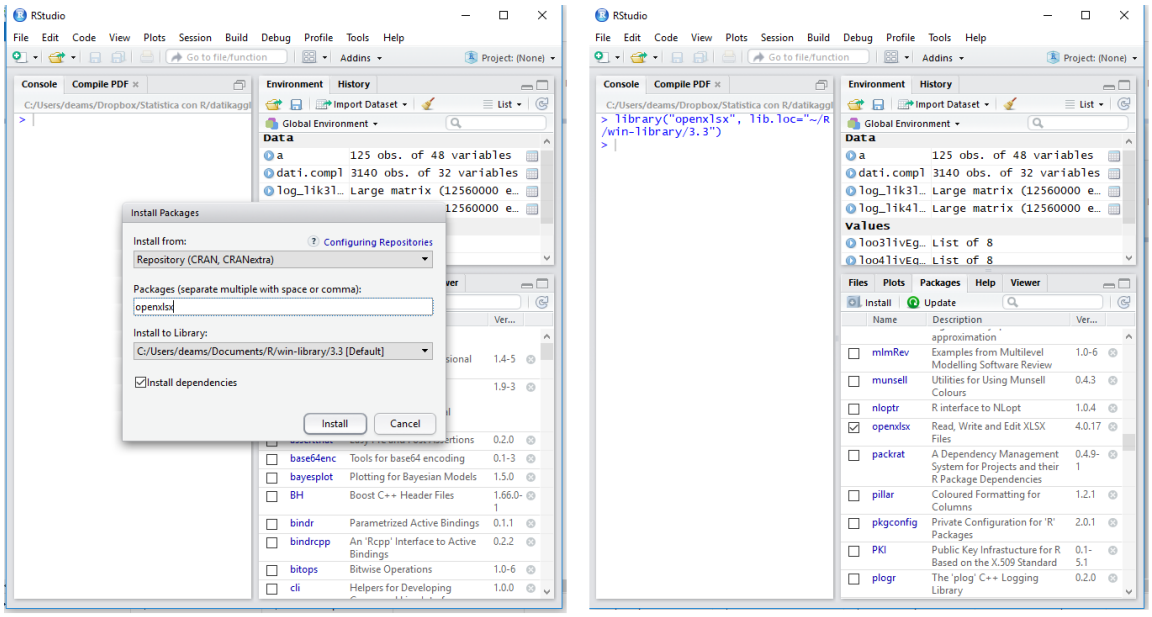


Figura 1.3: Installare e caricare pacchetti in R Studio.

Leggiamo il file nel modo seguente:

```
a = read.xlsx("../FoodChoices/food_coded_rev2.xlsx")
```

Attraverso tale comando il file `food_coded_rev2.xlsx` viene importato e automaticamente convertito nel data frame `a`. Si osservi che il percorso del file, se non salvato nella cartella di lavoro, viene scritto con lo "slash" /. Una volta importati i dati, per verificare la dimensione del data frame usiamo il comando `dim()`:

```
dim(a)

## [1] 125  48
```

Per visualizzare le prime righe del data frame possiamo usare

```
head(a)
```

oppure ricorrere all'apposita finestra "Environment" in alto a destra per osservarne la struttura e il contenuto. Se utilizzato su un data frame, il comando `length()` restituisce il numero delle variabili:

```
length(a)

## [1] 48
```

Infine il comando `names()` consente di accedere ai nomi delle variabili ed eventualmente modificarli. Si noti che oltre alle funzioni illustrate ne esistono molte altre tra cui, ad esempio, le funzioni `read.spss`, `read.dta` e `read.dbf` nel pacchetto `foreign` per leggere un file SPSS, Stata, e DBF, rispettivamente, la funzione `read.fwf` (fixed width files) nel pacchetto `utils`. Per ulteriori dettagli si può consultare il manuale base di R disponibile all'url https://cran.r-project.org/manuals.html interamente dedicato all'importazione di dati.

1.3.2 Manipolazione di data frame: primi esempi

Adesso che abbiamo importato un data frame vediamo alcuni comandi utili per indicizzare casi e/o variabili ed eseguire semplici operazioni. Per accedere al contenuto di una singola "variabile" del nostro data frame basterà usare il comando `<nome dataframe>[[i]]`, dove i denota la posizione corrispondente al vettore (variabile) desiderato. Questa modalità corrisponde ad uno dei possibili modi in cui è possibile accedere ai singoli elementi di un data frame o parte di essi. Per accedere, ad esempio, alla prima variabile abbiamo tre modi diversi, che implicano altrettanti modi di considerare un data frame:

1. data frame come `list`

   ```
   a[[1]]
   ```

2. data frame come `matrix`

   ```
   a[,1]
   ```

3. data frame come `data.frame`

```
a$GPA
```

Nel primo caso stiamo puntando al primo elemento (un vettore, nel nostro caso) nella lista; nel caso 2 stiamo chiedendo di estrarre la prima colonna, considerando tutte le righe nella "matrice" `a`; nel terzo caso stiamo invece chiedendo la restituzione della variabile `GPA`. Quest'ultimo caso rappresenta la modalità consigliabile.
I comandi per estrarre elementi da un vettore o una matrice possono essere utilizzati ed adattati anche al caso di estrazione di sottoinsiemi da data frame. Ad esempio, con il comando

```
a[2:7,2:6]

##   Gender breakfast calories_chicken calories_day calories_scone
## 2      1         1              610            3            420
## 3      1         1              720            4            420
## 4      1         1              430            3            420
## 5      1         1              720            2            420
## 6      1         1              610            3            980
## 7      2         1              610            3            420
```

estraiamo i casi dal 2 al 7 (quindi dalla riga 2 alla riga 7, inclusa) e le variabili dalla 2 alla 6; mentre con il seguente comando:

```
a[2:7,2]

## [1] 1 1 1 1 1 2
```

estraiamo gli stessi casi per la variabile 2 (`Gender`). Così facendo la parte del data frame estratta è considerata nella sua struttura originale e cioè come vettore numerico. Infatti, con il comando `class()` si ottiene

```
mode(a[2:7,2])

## [1] "numeric"
```

Si ottiene lo stesso risultato utilizzando il simbolo $ abbinato al nome della variabile:

```
a$Gender[2:7]

## [1] 1 1 1 1 1 2
```

Si noti che la variabile `Gender` è codificata come numerica, ma potrebbe essere utile trasformarla in fattore e definirne i livelli:

```
a$Gender=factor(a$Gender)
a$Gender

##   [1] 2 1 1 1 1 1 2 1 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 1 1 2 2 2 2
##  [31] 1 2 2 1 1 1 1 1 1 2 2 1 1 1 1 1 2 2 2 2 2 2 2 2 2 1 2 1 1 1
##  [61] 1 1 2 1 1 1 1 1 1 1 1 1 2 2 2 1 1 2 1 2 1 1 1 1 1 1 1 2 2 1
##  [91] 2 1 2 2 2 2 2 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 2
## [121] 1 1 1 2 1
## Levels: 1 2

levels(a$Gender)=c("Female","Male")
str(a$Gender)

##  Factor w/ 2 levels "Female","Male": 2 1 1 1 1 1 2 1 1 1 ...
```

Il comando

```
length(a$Gender[a$Gender=="Female"])

## [1] 76
```

o, equivalentemente

```
sum(a$Gender=="Female")
```

restituisce la frequenza assoluta della determinazione femmina per la variabile sesso. Come accennato precedentemente, la funzione `factor()` è particolarmente utile quando i dati contengono variabili categoriali con modalità espresse in forma numerica. Essa sostanzialmente trasforma i numeri in etichette (*levels*). consideriamo ora la variabile `calories_day` presenta le seguenti modalità:

1. I dont know how many calories I should consume
2. it is not at all important
3. it is moderately important
4. it is very important

Conviene ricodificarla come fattore e cambiare anche i nomi dei livelli in modo che siano più chiari:

```
a$calories_day=factor(a$calories_day)
levels(a$calories_day)  #N.B.: nessuna osservazione per il livello 1

## [1] "2" "3" "4"

newlev=c("not important", "moderately important", "very important")
levels(a$calories_day)=newlev
```

È importante notare che l'ordine dei livelli del fattore determinerà l'ordine di apparizione degli stessi nei grafici, tabelle e output di funzioni che utilizzano tale informazione.
Un altro modo per selezionare parti di data frame è rappresentato dall'uso di filtri. Ad esempio, se siamo interessati a filtrare la variabile `weight` (peso in libbre) solo per i maschi, possiamo utilizzare il seguente comando che richiede l'operatore `==`:

```
a[a$Gender=="Male", "weight"]

##  [1] 187 180 264 185 180 170 165 175 195 185 185 160 175 180 167
## [16] 205  NA 175 140 168 145 155 150 169 185 200 265 165 175 210
## [31] 140 200  NA 200 145 155 175 260 190 165 175 184 210 185 170
## [46] 138 150 185 135
```

o alternativamente

```
a$weight[a$Gender=="Male"]
```

Volendo estrarre la colonna `sports` (1=Yes, 2=No) per le donne, si userà il comando

```
a[a$Gender=="Female", "sports"]

##  [1]  1  2  2  1  2  2  2  1  1  1  1  2  2  2  1  1  2  1  1  1
## [21]  1  1  2  2  2  1  2  1  1  1  1  1  2  1  1  2  1  1  1  2
## [41]  1  1  2  1  1  2  1  1  1  1  2  1  2  2  1  1  1  2  2  2
## [61]  2  1  2  2  1  2  1  2  2  2  2  2  1 NA  2  2
```

Per estrarre alcuni valori all'interno di una variabile contenuta in un data frame vi sono quindi due modi: la selezione può essere effettuata sugli elementi del vettore estratto (ad es. `a$Gender`); oppure la condizione per selezionare i casi può essere specificata all'interno delle parentesi quadre prima della virgola, mentre dopo la virgola si specificano i nomi delle variabili a cui la condizione va applicata. Si può estendere il numero delle variabili da selezionare e complicare la condizione di inclusione (o esclusione) contemplando contemporaneamente anche altri simboli quali `&` (operatore logico *and*) e `|` (operatore logico *or*). Questi due operatori logici vengono solitamente utilizzati per richiedere che le condizioni si verifichino "simultaneamente" o "alternativamente".
Se ad esempio fossimo interessati al vettore `Gender` per gli studenti che fanno sport e fanno uso di vitamine potremmo usare il comando

```
a$Gender[a$sport==1 & a$vitamins==1]

##  [1] Male   Male   Female Female Male   Male   Male   Male
##  [9] Female Male   Female Female Male   Male   Male   Male
## [17] Female Female Female Male   Female Female Female Female
## [25] Male   Female Female Female Male   Male   Male   Male
## [33] Female Female
## Levels: Female Male
```

Invece una selezione un po' più complicata è la seguente: vogliamo selezionare i valori delle variabili `fruit_day` e `veggie_day` per le donne impegnate in una relazione che non praticano sport; quindi usiamo il comando:

```
a[a$Gender=="Female" & a$sport==2 & a$marital_status==2,
   c("fruit_day","veggies_day")]

##     fruit_day veggies_day
## 3           5           5
## 4           4           3
## 6           2           1
## 9           4           3
## 17          5           5
## 19          5           5
## 25          4           4
## 44          4           4
## 46          5           5
## 62          3           3
## 66          3           3
## 70          3           5
## 76          5           4
## 86          5           4
## 92          2           1
## 113         5           3
## 114         4           2
## 119         5           5
## 125         3           4
```

dove le variabili selezionate riguardano il consumo giornaliero di frutta e verdura, rispettivamente, e presentano le modalità:

1. very unlikely
2. unlikely
3. neutral
4. likely
5. very likely

Il comando `c(<oggetto1>, <oggetto2>, ...)` corrisponde all'operazione concatenazione di oggetti. La condizione di selezione dei record è sostanzialmente un vettore logico (ottenuto combinando altri vettori logici) che quando utilizzato come filtro fa in modo che solo gli elementi in corrispondenza di `TRUE` vengano selezionati.

La funzione `subset()` con argomenti `subset` e `select` rappresenta un modo più diretto attraverso cui specificare i casi e le variabili da includere (o escludere) nella selezione. Per ottenere lo stesso risultato dell'esempio precedente basterà digitare:

```
subset(a,subset=(Gender=="Female" & a$sport==2 & a$marital_status==2),
      select= c("fruit_day","veggies_day"))

##     fruit_day veggies_day
## 3           5           5
## 4           4           3
## 6           2           1
## 9           4           3
## 17          5           5
## 19          5           5
## 25          4           4
## 44          4           4
## 46          5           5
## 62          3           3
## 66          3           3
## 70          3           5
## 76          5           4
## 86          5           4
## 92          2           1
## 113         5           3
## 114         4           2
## 119         5           5
## 125         3           4
```

Si noti che è preferibile utilizzare la funzione `subset` per la selezione nel caso in cui siano presenti dati mancanti.
La selezione

```
subset(a, subset=(employment==1 | employment==2), select=-weight)
```

restituisce i valori di tutte le variabili esclusa la variabile `weight` (il segno "-" indica l'eliminazione della variabile dalla selezione) per gli studenti che lavorano a tempo pieno (`employment==1`) o a tempo parziale (`employment==2`).

La funzione principale per tabulare i dati presenti in un dataset è `table()`. Come vedremo questa funzione molto utile può essere utilizzata per ottenere la distribuzione di frequenze per una variabile, ma anche tabelle di contingenza. Il suo utilizzo più semplice consente di ottenere in forma di tabella il numero di casi corrispondente a ciascuna modalità di una variabile numerica o categoriale:

```
table(a$Gender)   #frequenze assolute

##
## Female   Male
##     76     49
```

Possiamo utilizzare la funzione `length()` per ottenere le frequenze relative della variabile `Gender`:

```
table(a$Gender)/length(a$Gender)   #frequenze relative

##
## Female   Male
##  0.608  0.392
```

Vediamo un altro esempio, in cui vogliamo definire come oggetto la tabella delle frequenze assolute per la variabile `diet_current_coded`, dopo averla ricodificata come fattore con opportuni livelli:

```
a$diet_current_coded=factor(a$diet_current_coded)
levels(a$diet_current_coded)=c("healthy/balanced/moderated",
         "unhealthy/cheap/too much/random",
         "the same thing over and over", "unclear")

tableDiet=table(a$diet_current_coded)
tableDiet

##
##      healthy/balanced/moderated unhealthy/cheap/too much/random
##                              50                              60
##    the same thing over and over                         unclear
##                              10                               5
```

Nel caso in cui le etichette che descrivono i nomi dei fattori siano lunghe (come in questo caso), possiamo associare il comando `cbind()` al comando `table` per avere un output più ordinato:

```
cbind(table(a$diet_current_coded))

##                                [,1]
## healthy/balanced/moderated       50
## unhealthy/cheap/too much/random  60
## the same thing over and over     10
## unclear                           5
```

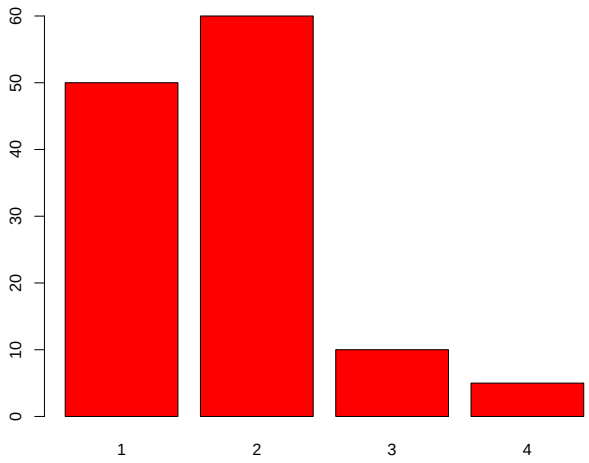
Usiamo il comando `margin.table`

```
margin.table(tableDiet)

## [1] 125
```

che restituisce banalmente il numero dei casi meno il numero dei valori mancanti (NA), se la variabile ne contiene. Possiamo visualizzare la distribuzione del tipo di dieta attraverso la funzione `barplot()`:

```
barplot(tableDiet, names.arg = c("1", "2", "3", "4"), col=2)
```

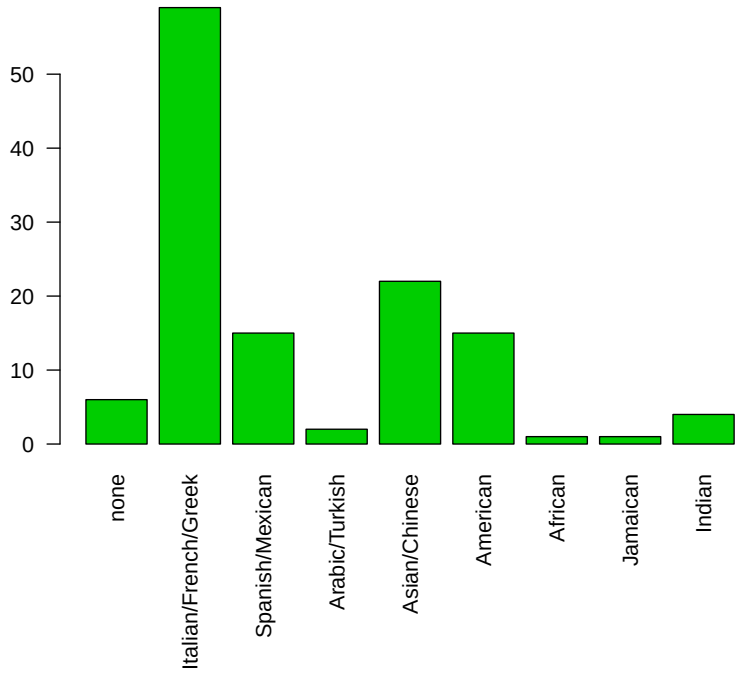


Procediamo analogamente per ottenere il grafico a barre della variabile che descrive il tipo di cucina preferita (`fav_cuisine_coded`):

```
a$fav_cuisine_coded=factor(a$fav_cuisine_coded)
levels(a$fav_cuisine_coded)=c("none","Italian/French/Greek",
                              "Spanish/Mexican","Arabic/Turkish",
                              "Asian/Chinese","American",
                              "African","Jamaican","Indian")
tableCuisine=table(a$fav_cuisine_coded)
tableCuisine

##
##                 none Italian/French/Greek
##                    6                   59
##      Spanish/Mexican       Arabic/Turkish
##                   15                    2
##        Asian/Chinese             American
##                   22                   15
##              African             Jamaican
##                    1                    1
##               Indian
##                    4

par(mar=c(9,4,4,4))  #increase margin
barplot(tableCuisine, las=2, col=3)  #grafico a barre
```



Il comando `table()` può essere utilizzato anche per costruire tabelle a due o più entrate. Gli esempi che seguono mostrano la costruzione di una tabella di contingenza con frequenze assolute, relative e percentuali. Si considerino le variabili `Gender` e `breakfast` (ai partecipanti al sondaggio viene chiesto di associare una figura raffigurante dei cereali e una raffigurante delle ciambelle alla parola "breakfast"); definiamo quindi tale variabile come fattore:

```
a$breakfast=factor(a$breakfast)
levels(a$breakfast)=c("cereal","donut")
```

Usiamo il comando `table()` per creare una tabella di contingenza con la prima variabile messa in riga e la seconda in colonna, e successivamente `prop.table()` per ottenere le frequenze relative congiunte

```
tableBreak=table(a$Gender,a$breakfast) #creo la tabella
tableBreak
```

```
##
##          cereal donut
##   Female     70     6
##   Male       41     8
```

```
prop.table(tableBreak)                #uso il comando prop.table()
```

```
##
##          cereal donut
##   Female  0.560 0.048
##   Male    0.328 0.064
```

Per condizionare rispetto al sesso

```
prop.table(tableBreak, margin=1)

##
##              cereal      donut
##   Female 0.92105263 0.07894737
##   Male   0.83673469 0.16326531

round(prop.table(tableBreak, margin=1), 2)

##
##          cereal donut
##   Female   0.92  0.08
##   Male     0.84  0.16
```

si noti che con l'opzione `margin` si specifica la variabile rispetto alla quale si vuole condizionare, mentre `round()` è utilizzato per specificare il numero di decimali desiderato per i valori nella tabella, 2 in questo caso.
È possibile selezionare le modalità rispetto alle quali visualizzare la tabella delle frequenze. Se, ad esempio, vogliamo ottenere la distribuzione di coloro che dichiarano di seguire una dieta sana, o moderatamente bilanciata, per sesso digitiamo i seguenti comandi:

```
tableHealthy=table(a$diet_current_coded=="healthy/balanced/moderated",
             a$Gender)
tableHealthy

##
##         Female Male
##   FALSE     41   34
##   TRUE      35   15

round(prop.table(tableHealthy, margin = 2), 2)

##
##         Female Male
##   FALSE   0.54 0.69
##   TRUE    0.46 0.31
```

La funzione `cbind()` può essere usata per aggiungere colonne al nostro data frame. Ad esempio, se vogliamo aggiungere una colonna avente un numero identificativo per ciascun caso creiamo il vettore `id`:

```
id=1:nrow(a)       #creo vettore con identificativo

a=cbind(id,a)      #aggiungo il vettore al data frame
head(a)
```

`cbind` può essere utilizzata anche per aggiungere nuove variabili create da quelle originali. Ad esempio, immaginiamo di voler esprimere il peso (in libbre) in chilogrammi:

```
kg=round(a$weight/2.2046, 2)
a=cbind(a, kg)
head(a)
```

In alternativa, si può utilizzare il comando:

```
a$kg=a$weight/2.2046
```

La funzione `rbind()` può essere utilizzata in modo analogo per operare sulle righe del data frame.

1.3.3 Trattamento dei dati mancanti

Molto spesso i dati raccolti (soprattutto da questionario) sono caratterizzati da valori mancanti che in R sono etichettati con la stringa `NA`. Se si importano dati da un file esterno è molto probabile che i dati mancanti siano etichettati con qualcosa di diverso da `NA`; ad esempio un asterisco, una cella vuota oppure valori numerici implausibili, come 9999. In quest'ultimo caso, ad esempio, nel data frame tutti i 9999 dovranno essere sostituiti da un `NA`. Questo può essere ottenuto facilmente utilizzando l'argomento `na.strings=999` nella funzione `read.table()`, oppure una sostituzione può avvenire facilmente nel modo seguente:

```
<dataframe>[<dataframe> == 999] = NA
```

Una funzione molto utile per individuare i dati mancanti è `is.na()` che restituisce il valore logico `TRUE` se il dato è mancante:

```
is.na(a)
```

Alcune funzioni in R hanno speciali argomenti per trattare i dati mancanti. L'opzione `na.rm=TRUE` esegue la funzione escludendo i dati mancanti. Ad esempio, supponiamo di voler calcolare il numero di studenti americani con un *Grade Point Average* (GPA) inferiore a 3.5:

```
sum(a$GPA<3.5)

## [1] NA

sum(a$GPA<3.5, na.rm=TRUE)

## [1] 57
```

Se invece vogliamo calcolare il peso medio

```
mean(a$kg)

## [1] NA

mean(a$kg, na.rm=TRUE)

## [1] 71.84116
```

Tuttavia, alcune funzioni non dispongono di alcun parametro per i dati mancanti. Pertanto, riconoscere i dati mancanti risulta fondamentale. Ad esempio, è possibile selezionare a priori solo i dati non-mancanti utilizzando la stessa funzione `is.na()`. Volendo selezionare dal nostro data frame i record con dati non-mancanti nella variabile `weight`:

```
a1<-a[!is.na(a$weight),]
head(a1)
```

Possiamo ad esempio creare il vettore `pesomaschi` usando `a1` e determinare il numero di maschi che pesano più di 90 kg:

```
pesomaschi=a1$kg[a1$Gender=="Male"]

#frequenze (relative) condizionate
sum(pesomaschi>90)/length(pesomaschi)

## [1] 0.1914894
```

La funzione `na.omit()` restituisce il data frame senza le righe che contengono valori mancanti.

```
na.omit(a)
```

1.3.4 Un esempio reale (più complesso)

Consideriamo ora i dati raccolti nel 2016 da Democracy Fund Voter Study Group (https://www.voterstudygroup.org/) e riguardanti l'atteggiamento degli elettori americani nei confronti di diversi categorie di individui. I dati sono stati estratti da un data set più complesso e salvati come file di estensione `.RData`. Carichiamo quindi il *workspace* contenente i dati dalla voce del menu 'Session' > 'Load Workspace', scegliendo il file "feeling.Rdata". Il data frame, che ora è disponibile nel workspace, si presenta composto da 8000 osservazioni e 15 variabili tutte numeriche che esprimono un punteggio da 0 a 100. In particolare, i punteggi da 0 a 50 indicano un atteggiamento non favorevole o indifferente (50) verso i gruppi di persone oggetto

del quesito (ad esempio, immigrati, minoranze etniche e religiose), mentre i punteggi da 51 a 100 indicano un atteggiamento favorevole.

Supponiamo di voler calcolare il punteggio medio per tutte le 15 variabili che compongono il data frame. Possiamo utilizzare la funzione `mean`. Per evitare di dover digitare lo stesso comando per ciascuna variabile, possiamo rendere 'automatica' questa operazione utilizzando un ciclo `for`. La sintassi è la seguente:

```
for (i in expr1) {expr2}
```

dove `i` è l'indice del ciclo, `expr1` è il vettore di valori per `i` e `expr2` è l'espressione che deve essere ripetuta tante volte quanti sono gli elementi di `expr1`. Nel nostro caso

```
for (i in 1:length(feeling))
  {
  print(mean(feeling[, i], na.rm=TRUE))
  }
```

```
## [1] 71.86483
## [1] 75.2162
## [1] 70.48553
## [1] 73.8897
## [1] 51.09848
## [1] 77.232
## [1] 73.30804
## [1] 52.47108
## [1] 59.39763
## [1] 41.16396
## [1] 32.56286
## [1] 63.06448
## [1] 51.58139
## [1] 77.95412
## [1] 28.44089
```

Lo stesso risultato si può ottenere in maniera più efficiente attraverso la funzione `apply`, a condizione che le variabili siano dello stesso tipo (es. tutte numeriche).

```
Mean=apply(feeling, 2, mean, na.rm=TRUE)
Mean
```

```
##    ft_black_2016    ft_white_2016     ft_hisp_2016
##          71.86483         75.21620         70.48553
##    ft_asian_2016   ft_muslim_2016      ft_jew_2016
##          73.88970         51.09848         77.23200
##   ft_christ_2016      ft_fem_2016    ft_immig_2016
```

```
##         73.30804         52.47108         59.39763
##      ft_blm_2016   ft_wallst_2016     ft_gays_2016
##         41.16396         32.56286         63.06448
##   ft_unions_2016   ft_police_2016 ft_altright_2016
##         51.58139         77.95412         28.44089
```

Si noti che R consente di ottenere facilmente le principali statistiche sulle variabili del data set mediante il comando `summary()`:

```
summary(feeling)    #riassunti statistici

##  ft_black_2016    ft_white_2016     ft_hisp_2016
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.: 52.00   1st Qu.: 57.00   1st Qu.: 51.00
##  Median : 77.00   Median : 80.00   Median : 75.00
##  Mean   : 71.86   Mean   : 75.22   Mean   : 70.49
##  3rd Qu.: 91.00   3rd Qu.: 93.00   3rd Qu.: 90.00
##  Max.   :100.00   Max.   :100.00   Max.   :100.00
##  NA's   :232      NA's   :220      NA's   :225
##  ft_asian_2016    ft_muslim_2016   ft_jew_2016
##  Min.   :  0.00   Min.   :  0.0   Min.   :  0.00
##  1st Qu.: 55.00   1st Qu.: 26.0   1st Qu.: 60.00
##  Median : 79.00   Median : 51.0   Median : 81.00
##  Mean   : 73.89   Mean   : 51.1   Mean   : 77.23
##  3rd Qu.: 91.00   3rd Qu.: 77.0   3rd Qu.: 96.00
##  Max.   :100.00   Max.   :100.0   Max.   :100.00
##  NA's   :239      NA's   :425     NA's   :237
##  ft_christ_2016    ft_fem_2016     ft_immig_2016
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.0
##  1st Qu.: 51.00   1st Qu.: 26.00   1st Qu.: 46.0
##  Median : 81.00   Median : 51.00   Median : 60.0
##  Mean   : 73.31   Mean   : 52.47   Mean   : 59.4
##  3rd Qu.: 98.00   3rd Qu.: 80.00   3rd Qu.: 80.0
##  Max.   :100.00   Max.   :100.00   Max.   :100.0
##  NA's   :212      NA's   :479      NA's   :312
##   ft_blm_2016     ft_wallst_2016    ft_gays_2016
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.:  4.00   1st Qu.: 10.00   1st Qu.: 50.00
##  Median : 39.00   Median : 30.00   Median : 66.00
##  Mean   : 41.16   Mean   : 32.56   Mean   : 63.06
##  3rd Qu.: 75.00   3rd Qu.: 50.00   3rd Qu.: 90.00
##  Max.   :100.00   Max.   :100.00   Max.   :100.00
##  NA's   :620      NA's   :579      NA's   :308
##  ft_unions_2016   ft_police_2016   ft_altright_2016
##  Min.   :  0.00   Min.   :  0.00   Min.   :  0.00
##  1st Qu.: 25.00   1st Qu.: 66.00   1st Qu.:  2.00
```

```
##  Median : 50.50   Median : 86.00   Median : 19.00
##  Mean   : 51.58   Mean   : 77.95   Mean   : 28.44
##  3rd Qu.: 80.00   3rd Qu.: 97.00   3rd Qu.: 50.00
##  Max.   :100.00   Max.   :100.00   Max.   :100.00
##  NA's   :468      NA's   :175      NA's   :1867
```

1.3.5 Ricodifica di una variabile quantitativa in classi

A volte può essere necessario "categorizzare" in classi una variabile quantitativa. Nel nostro caso, nel data frame `feeling` le variabili si prestano a tale trasformazione. Ad esempio proviamo a trasformare la variabile `ft_immig_2016` in una variabile categoriale con quattro classi: per nulla favorevole (0-25), non favorevole o indifferente (26-50), mediamente favorevole (51-75), molto favorevole (76-100).
Utilizziamo la funzione `cut()`, avendo l'accortezza di inserire l'estremo della prima classe con un valore inferiore al minimo osservato (nel nostro caso possiamo usare -1):

```
ImmCl=cut(feeling$ft_immig_2016, breaks=c(-1, 25, 50, 75, 100),
                  labels=c("Per nulla", "Basso", "Medio", "Alto"))

feeling$Immig2=factor(ImmCl) #aggiunge la colonna al data frame
```

In `breaks` possono essere espressi sia gli estremi degli intervalli (come nell'esempio sopra) o il numero degli intervalli desiderato, mentre `labels` serve soltanto a nominare le etichette della variabile categoriale appena creata; ponendo `labels=FALSE` i semplici interi $1, 2, \ldots$ vengono utilizzati come etichette.

1.3.6 Esportazione dei dati

Un data frame può essere facilmente salvato come file. Ad esempio, se vogliamo esportare dati in formato `.csv` possiamo utilizzare il comando `write.table()` con opportuni argomenti:

```
write.table(feeling,file="feeling2.csv", sep=",", row.names = FALSE)
```

Si rimanda all'*help* per la sintassi delle funzioni `write.csv()` e `write.csv2()`.

Capitolo 2

Rappresentazioni grafiche e prime analisi statistiche in R

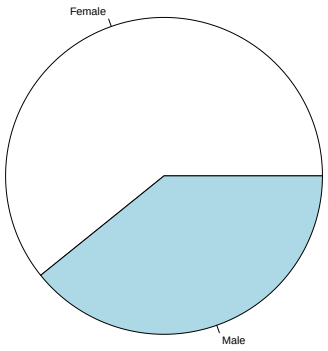
2.1 Tipi di variabili

In generale, le variabili possono essere classificate in qualitative (o categoriali) e quantitative (o numeriche). Una variabile qualitativa assegna i casi a categorie; essa può essere a sua volta classificata in nominale o ordinale. Una variabile quantitativa ha valori che descrivono una quantità misurabile con un numero; essa può essere a sua volta classificata in discreta o continua.
Questa classificazione è utile poiché è possibile effettuare diverse analisi statistiche a seconda del tipo di variabile da analizzare.

2.1.1 Variabili categoriali

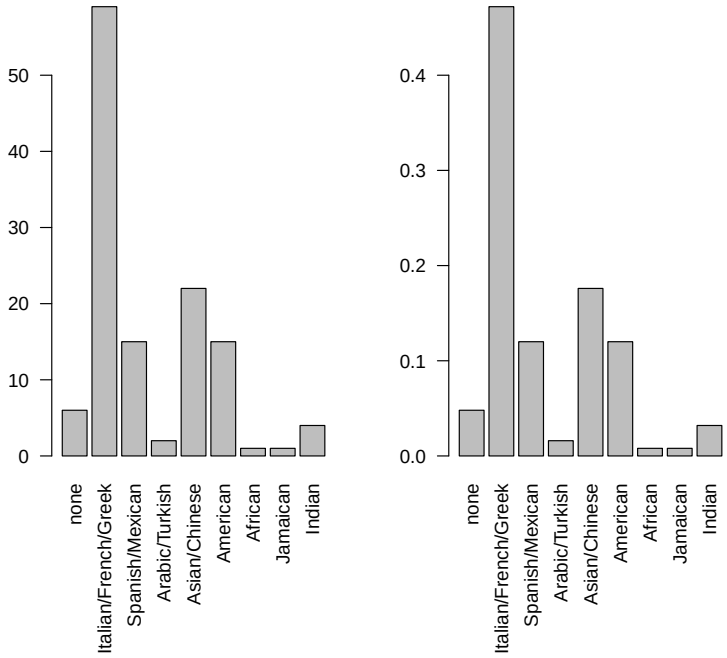
Consideriamo il data frame riguardanti le abitudini alimentari di 125 studenti americani, che in precedenza abbiamo chiamato `a`.
È possibile rappresentare graficamente la variabile qualitativa `Gender` con un diagramma a torta (piechart), mediante il comando:

```
pie(table(a$Gender))
```



La variabile `fav_cuisine_coded` può essere rappresentata con un diagramma a barre, con frequenze assolute o relative:

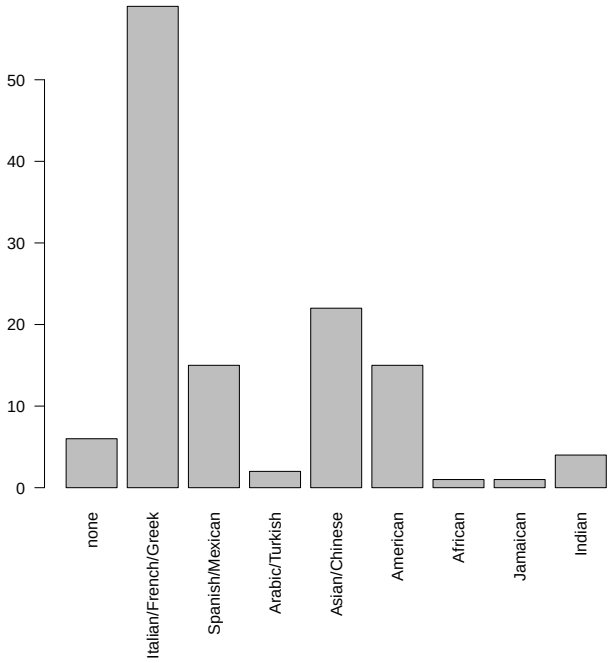
```
par(mfrow=c(1,2),las=1, mar=c(9,4,1,1))
barplot(table(a$fav_cuisine_coded), las = 2)
barplot(table(a$fav_cuisine_coded)/length(a$fav_cuisine_coded), las=2)
```



Si noti che la funzione `par` consente di controllare diversi parametri grafici, in modo tale da personalizzare l'aspetto dei grafici secondo le proprie esigenze (si consulti l'*help* per maggiori dettagli). In particolare `par(mfrow=c(i,j))` è utile nel caso in cui si vogliano visualizzare più grafici contemporaneamente e divide la finestra grafica in `i` righe e `j` colonne.

Si può ottenere lo stesso risultato col comando `plot()`

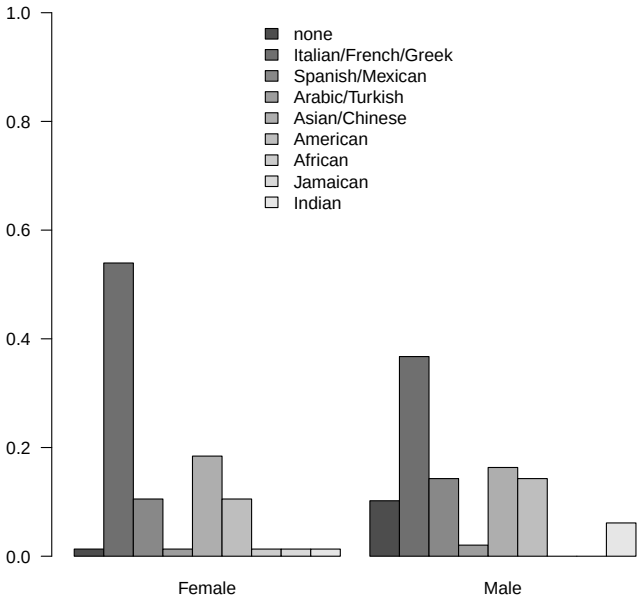
```
par(mfrow=c(1,1), mar=c(9,4,1,1))
plot(a$fav_cuisine_coded, las=2)
```



Le distribuzioni condizionate rispetto alla variabile `Gender` possono essere visualizzate con i comandi

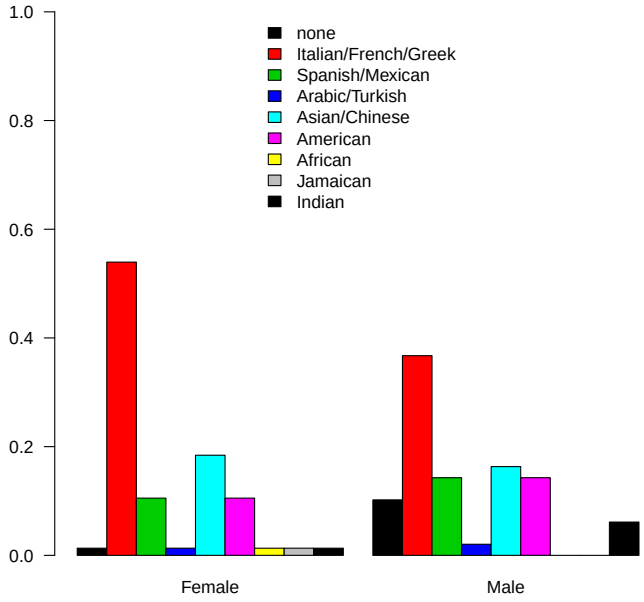
```
par(mfrow=c(1,1),las=1)
tab.cond=table(a$fav_cuisine_coded, a$Gender)

barplot(prop.table(tab.cond,2),beside=T,legend=T,ylim=c(0,1.),
        args.legend=list(x ="top",bty="n"))
```



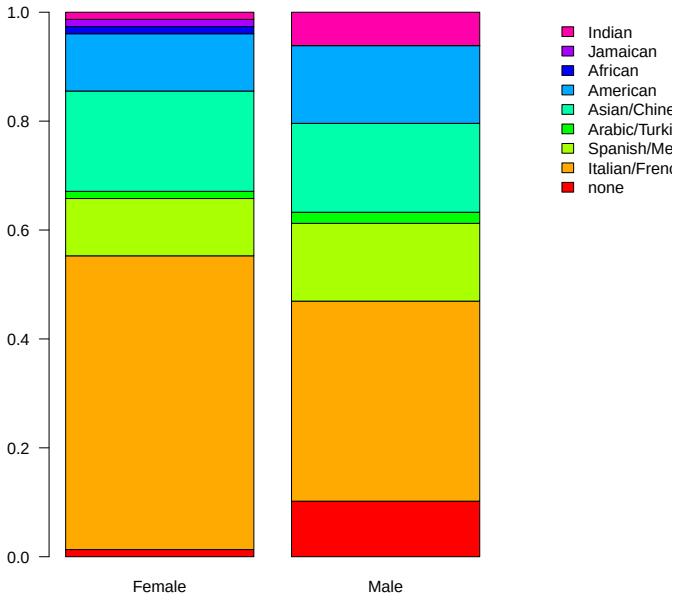
Se si vuole aggiungere i colori,

```
par(mfrow=c(1,1),las=1)
barplot(prop.table(tab.cond,2), beside=T, legend=T, ylim=c(0,1.),
        args.legend=list(x ="top",bty="n"), col=1:9)
```



Per consultare la lista di colori disponibili in R si veda `http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf`. Si possono anche usare le palette di colori già presenti in R come `rainbow()` o `heat.colors()`. Se si vuole visualizzare un grafico con colonne in pila (*stacked bar chart*) si userà la stessa funzione con l'opzione predefinita `beside=FALSE`:

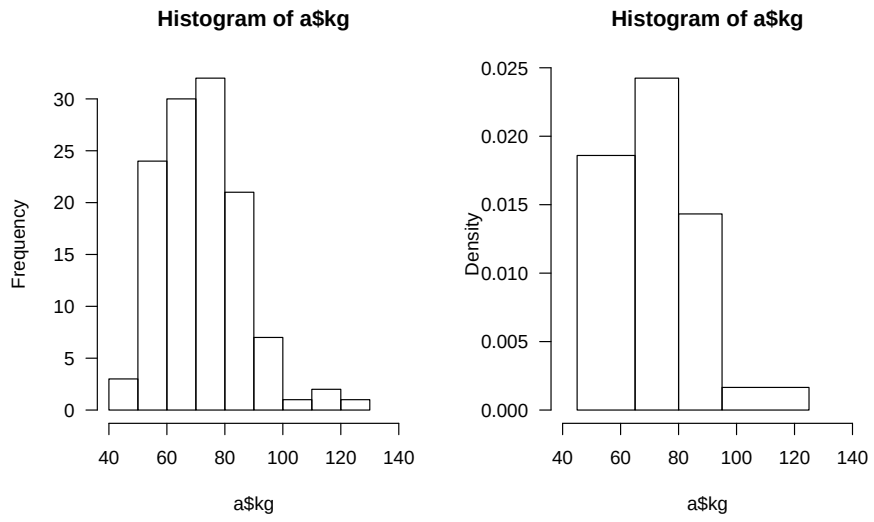
```
colors=rainbow(9)          #definizione colori
barplot(prop.table(tab.cond, 2), legend=T, ylim=c(0, 1),
       args.legend=list(x ="topright", bty="n", inset=c(-0.55, 0)),
       col=colors)
```



2.1.2 Variabili numeriche

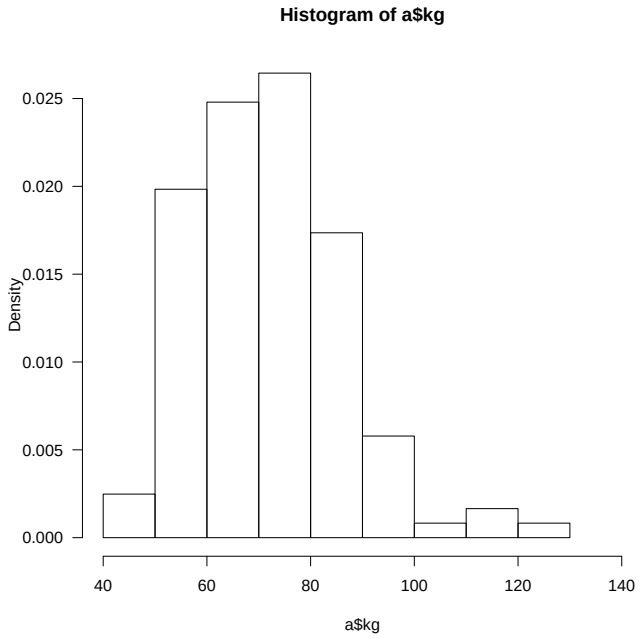
La distribuzione di una variabile numerica può essere rappresentata mediante un istogramma. È possibile cambiare l'ampiezza e/o il numero di intervalli tramite l'opzione `breaks`.

```
par(mfrow=c(1,2),las=1)
hist(a$kg, xlim = c(40,140))
hist(a$kg, breaks=c(45,65,80,95,125), xlim = c(40,140))
```



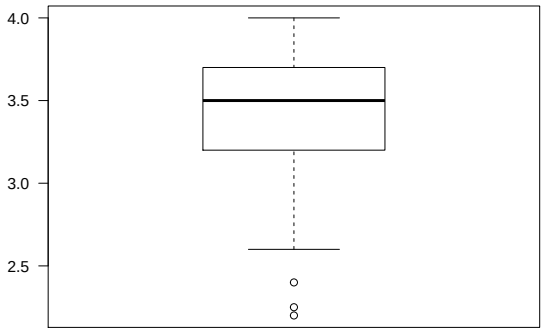
Se si vuole fare un confronto, può essere d'aiuto considerare le frequenze relative

```
par(mfrow=c(1,1),las=1)
hist(a$kg,freq=FALSE,xlim = c(40,140))
```



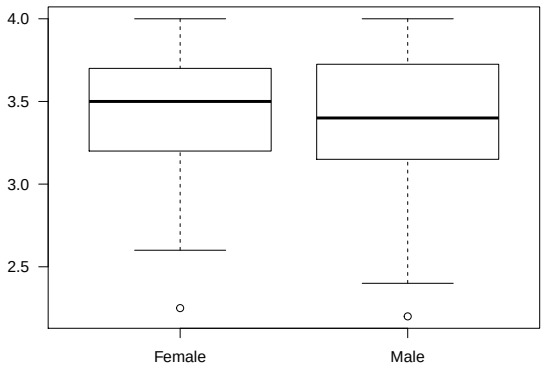
Per ottenere un diagramma a scatola e baffi (box plot) a partire da una variabile quantitativa (ad esempio, il `GPA`) si possono utilizzare i seguenti comandi

```
boxplot(a$GPA)
```



Ricordiamo che la scatola corrisponde alla parte centrale della distribuzione che comprende il 50% dei casi (dal primo al terzo quartile), la mediana è individuata dalla linea orizzontale nella scatola e i baffi corrispondono alle code di sinistra fino al primo quartile e di destra, dal terzo quartile in poi. I punti sul grafico fuori dai baffi rappresentano gli outliers.
Volendo ottenere diversi box plot per una stessa variabile quantitativa, suddividendo le osservazioni in base ai livelli di un fattore (ad esempio `GPA` rispetto a `Gender`) usiamo:

```
boxplot(a$GPA~a$Gender)
```



o, alternativamente

```
plot(a$Gender, a$GPA)
```

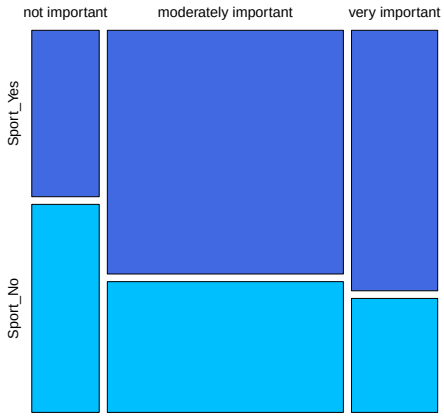
2.1.3 Rappresentazione di due o più variabili

Consideriamo il data frame `a` e le due variabili qualitative `sports` e `calories_day` che descrivono l'abitudine a praticare uno sport e l'attenzione verso le calorie consumate in un giorno. Conviene ricodificare i livelli della variabile `sports` come segue:

```
a$sports=factor(a$sports)
levels(a$sports)=c("Sport_Yes", "Sport_No")
```

Il grafico a mosaico è utile se si vogliono rappresentare più variabili qualitative insieme

```
tab.sc=table(a$calories_day, a$sports)
mosaicplot(tab.sc, color = c("royalblue", "deepskyblue"),
           main = " ", cex=1.1)
```

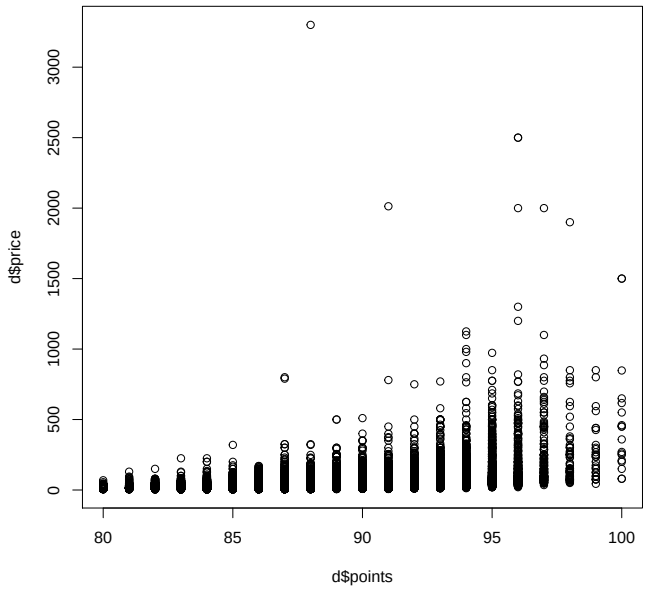


Possiamo usare il comando `plot()` per ottenere il diagramma di dispersione di due variabili numeriche. Vediamo un esempio considerando i dati salvati nel file `wine.Rdata`.

```
load("../Wine/wine.Rdata")
```

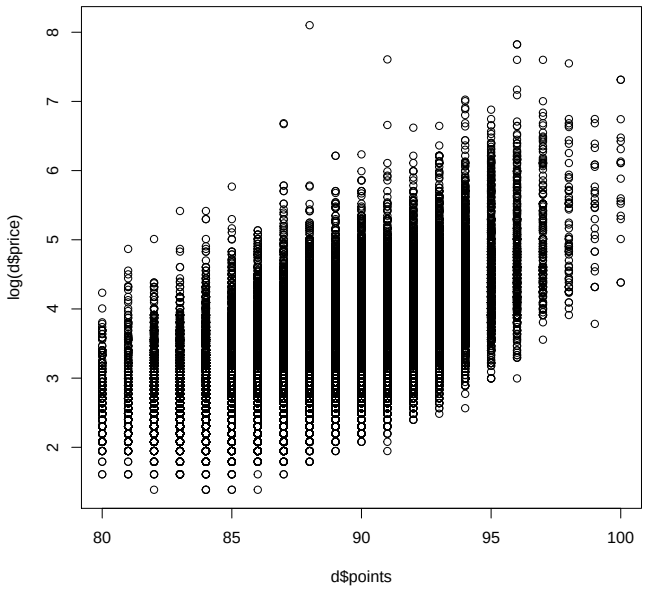
Il file riporta per 125335 vini il prezzo, il punteggio di gradimento e l'anno di produzione. Otteniamo il diagramma di dispersione tra punteggio di gradimento e prezzo con

```
plot(d$points, d$price)
```



Per la variabile `price` appare ragionevole usare la trasformazione `log(price)`:

```
plot(d$points,log(d$price))
```



Si vuole investigare la relazione tra le due quantità, per cui possiamo stimare un modello di regressione lineare usando la funzione `lm()`; nel suo uso più elementare scriviamo

```
fit=lm(log(d$price)~d$points)
```

questo crea un oggetto di tipo *lm* che contiene varie quantità relative alla stima del modello di regressione semplice avente `log(price)` come variabile risposta e `points` come variabile esplicativa:

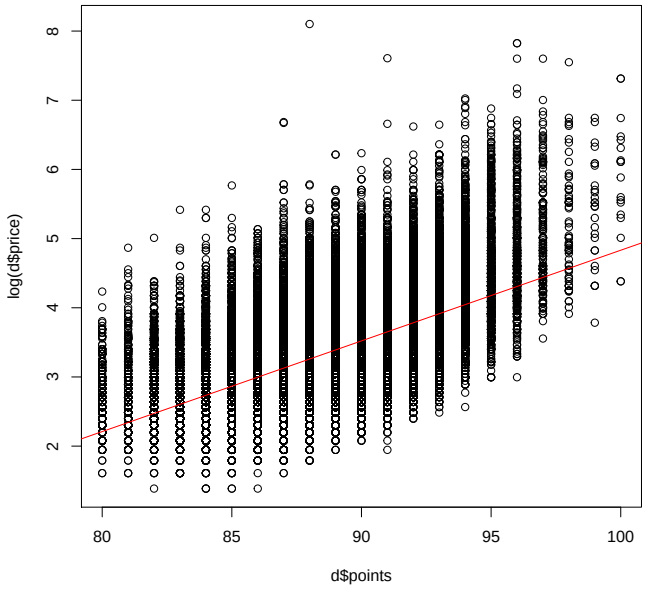
```
summary(fit)

##
## Call:
## lm(formula = log(d$price) ~ d$points)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7062 -0.3685 -0.0400  0.3124  4.8428
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.275511   0.043992  -188.1   <2e-16 ***
## d$points     0.131072   0.000497   263.7   <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.5173 on 116819 degrees of freedom
##   (8520 observations deleted due to missingness)
## Multiple R-squared:  0.3732,Adjusted R-squared:  0.3732
## F-statistic: 6.954e+04 on 1 and 116819 DF,  p-value: < 2.2e-16
```

La retta stimata può essere aggiunta al grafico con il comando `abline()`

```
plot(d$points,log(d$price))
abline(coef(fit), col=2)
```



Volendo includere anche la variabile `anno` il modello si può adattare usando il comando `lm()` già illustrato con una formula che comprende entrambe le esplicative:

```
fit2=lm(log(d$price)~d$points+d$anno)
summary(fit2)

##
## Call:
## lm(formula = log(d$price) ~ d$points + d$anno)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.7493 -0.3605 -0.0430  0.3159  4.9035
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 44.5743452  0.8180184   54.49   <2e-16 ***
## d$points     0.1331091  0.0004894  272.00   <2e-16 ***
```